Atty Docket No.: 50037.0188US01 / 303816.01

SYSTEM AND METHOD FOR DYNAMICALLY ALLOCATING RESOURCES IN A CLIENT/SERVER ENVIRONMENT

Field of the Invention

The present invention relates to network computing environments. More particularly, the present invention relates to server resource allocation in a network computing environment.

Background of the Invention

Client/server network environments have become ubiquitous. It is routine today for client computing systems to connect over a network to a server. Competition among network hardware and software providers drives them to offer more and more sophisticated products. One particular feature of network computing systems that is a point of competition is file system performance. Computing system developers strive to provide network servers with robust file system performance.

One limitation on server performance is the physical limitation of available server resources, such as hard disk space, processor power, and available memory. The amount of available memory can be particularly important in situations where multiple clients connect to a server simultaneously, as frequently occurs. More specifically, when multiple clients connect to the server, the server must allocate its resources to each of the clients. One example is receive buffers used to support file system transactions between the clients and the server. Every receive buffer allocated to a particular client consumes a certain amount of memory. In addition, different clients vary in their respective need for available server resources.

When the available resources exceeds the number of resources being requested by all the clients, there should be no noticeable performance impact on any one client because any one client could simple attempt to negotiate more resources. However, as happens with increasing frequency today, sometimes the number of resources being requested by the clients exceeds the available resources. Today, that situation is handled by simply refusing to accept new connections. In other words, once a client has been

allocated buffers or other resources, those resources remain allocated until the client releases them.

While this is one solution to the problem, the inventors have determined that this is an inequitable solution because in many cases, clients may have allocated but unused resources. If so, it seems unfair that a particular client with a disproportionate number of allocated server resources can continue to hold those resources to the exclusion of new clients, even if that client isn't making use of them.

An adequate mechanism for allocating server resources to multiple clients in an equitable manner has eluded those skilled in the art.

Summary of the Invention

The present invention is directed at a mechanism for equitably rebalancing server resources that are allocated to a plurality of clients connected to the server. Briefly stated, a client maintains information about how many transaction credits it has allocated and in use. When issuing a transaction request, the client sends a hint about how many additional resources the client may need to satisfy its current workload. The hint may take the form of a number of pending transaction requests that are currently queued at the client. The server analyzes the client's information about the additional needed resources and compares that information to a number of available resources at the server. The available resources may have been allocated to other clients connected to the server. If sufficient resources exists to satisfy the client's need, then those resources are allocated to the client. If sufficient resources do not exist, the server invokes a rebalancing mechanism to quantify an equitable number of resources that should be allocated to each client connected to the server. The server then issues rebalancing messages to any affected client to either reduce or increase the affected client's credit limit so that each client has an equitable allocation of resources.

Brief Description of the Drawings

FIGURE 1 is a functional block diagram that illustrates a computing device that may be used in implementations of the present invention.

FIGURE 2 is a functional block diagram generally illustrating a network environment in which embodiments of the invention may be implemented.

FIGURE 3 is a graphical representation of relevant components of a client computing system that stores resource allocation information in accordance with the invention.

FIGURE 4 is a pseudo-code representation of an illustrative process performed at the client for interacting with the server to achieve dynamic rebalancing of the client's allocated resources.

FIGURE 5 is a functional block diagram generally illustrating a server configured for use in the network environment of FIGURE 2.

FIGURE 6 is a pseudo-code representation of an illustrative process performed at the server for interacting with the client to achieve dynamic rebalancing of the client's allocated resources.

Detailed Description of the Preferred Embodiment

The invention will be described here first with reference to one example of an illustrative computing environment in which embodiments of the invention can be implemented. Next, a detailed example of one specific implementation of the invention will be described. Alternative implementations may also be included with respect to certain details of the specific implementation. It will be appreciated that embodiments of the invention are not limited to those described here.

Illustrative Computing Environment of the Invention

FIGURE 1 illustrates a computing device that may be used in illustrative implementations of the present invention. With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device **100**. In a very basic configuration, computing device **100** typically includes at least one processing unit **102** and system memory **104**. Depending on the exact configuration and type of computing device, system memory **104** may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory **104** typically includes an operating system **105**, one or more program

modules **106**, and may include program data **107**. This basic configuration of computing device **100** is illustrated in FIGURE 1 by those components within dashed line **108**.

Computing device **100** may have additional features or functionality. For example, computing device **100** may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage **109** and non-removable storage **110**. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory **104**, removable storage **109** and non-removable storage **110** are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks ("DVD") or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device **100**. Any such computer storage media may be part of device **100**. Computing device **100** may also have input device(s) **112** such as keyboard **122**, mouse **123**, pen, voice input device, touch input device, scanner, etc. Output device(s) **114** such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here.

Computing device **100** may also contain communication connections **116** that allow the device to communicate with other computing devices **118**, such as over a network. Communication connections **116** is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF,

infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

Discussion of Specific Implementation

FIGURE 2 is a functional block diagram generally illustrating a network environment in which embodiments of the invention may be implemented. As shown in FIGURE 2, a network environment **200** includes a server **201** and a plurality of clients (e.g., client **203, 205, 207**) connected to a network. Certain aspects of a representative client, such as client **203**, are described in detail below in conjunction with FIGURE 3. Certain aspects of the server **201** are described in detail below in conjunction with FIGURE 4. However, those devices are described here generally regarding the manner in which they interact and their respective roles in this implementation of the invention.

Each client has a connection (e.g., connections **204**, **206**, **208**, respectively) to the server **201**. The clients and the server **201** may communicate using one of many different communication protocols. One communication protocol that may be used for distributed file systems is the Light Weight I/O (LWIO) protocol. The LWIO protocol enables an application operating on one computer (i.e., the client **203**) to communicate directly with a file system on the server **201** without necessarily involving kernel-mode resources on the client computer. Bypassing kernel-mode operations reduces the overhead associated with distributed file access, resulting in improved performance over other protocols, like TCP/IP. Clients can, however, have both user and kernel level components performing file I/O transactions on the server **201**.

When a client, such as client **203**, initiates a transaction over the network, the server **201** allocates certain resources to handling the transaction for the particular client. Most notably, the server **201** allocates to the client a certain number of buffers **209** within its available system memory **210**. The buffers **209** are used to support I/O transactions from the client to the server **201**. Each time another client, such as client **205**, makes a connection to the server **201**, more of the server's resources are allocated to that new client. However, as noted above, the resources that the server **201** can make available are limited.

For the purpose of this discussion, each resource (e.g., receive buffers **209**) that the server **201** allocates to a client is termed a "transaction request credit" or simply a "credit." In accordance with the invention, the number of credits allocated by the server **201** to each client is dynamic, and may be modified over the lifetime of the several connections made by the several clients that connect to the server **201**. In addition, for the purpose of this discussion, the term "transaction" includes any access, such as a read or write, of data on the server **201**. As will be described in great detail below in conjunction with FIGUREs 3-6, the clients and the server **201** each maintain and share information that allows the server **201** to rebalance credit allocation in an equitable fashion that seeks to improve overall system performance. Each client, such as client **203**, maintains information (e.g., information **213**) about the state of its allocated credits and pending transactions. The client-side information **213** is illustrated in detail in FIGURE 3 and described below. Each client shares some of its respective information with the server **201**. The server **201** maintains global information **221** about the state of each client's allocated credits. The server-side information **221** is illustrated in detail in FIGURE 5 and described below. In a resource constrained situation, the server **201** uses the information from the clients to reallocate the server's resources among the several competing clients.

The following discussion describes in detail the interaction of the client and the server to dynamically allocate resources. First, the client will be described including the information stored on the client and how the client handles messages issuing transaction requests to the server. Next, the server will be described including the information stored on the server and how the server identifies and rectifies a resource constrained situation.

Illustrative Client Operation

FIGURE 3 is a functional block diagram generally illustrating an illustrative client **203** configured for use in the network environment of FIGURE 2. As above, the client **203** has a connection **204** to the server, and maintains information **213** used in connection with the rebalancing mechanisms of the present invention. The client **203** stores the following information for its connection to the server (1) the current

number of outstanding transaction requests (Credits Used), (2) the maximum number of credits or outstanding requests available (Credit Limit), (3) the number of requests that cannot be sent because the Credits Used equals the Credit Limit (Pending Count), and (4) any unset requests (Pending Queue). The Pending Count is also the length of the Pending Queue. The Pending Queue is processed in FIFO order with the possible exception that credit messages are inserted at the head of the Pending Queue. The client-side information **213** may also include additional information, such as status flags or the like.

FIGURE 4 is a pseudo-code representation of an illustrative process performed at the client **203** for interacting with the server **201** to achieve dynamic rebalancing of the client's allocated resources. The essential functionality of this pseudo-code could be implemented in software components executing on the client **203**.

Initially, when the client **203** connects to the server **201**, the two systems negotiate an initial allocation of transaction request credits. Once the negotiation of credits is complete, the client **203** is allowed to send a number of transaction requests up to the negotiated limit. A typical minimum number of credits could be two. One credit could be reserved by the client **203** to handle priority messages. In the case of LWIO one credit may be reserved for INTERRUPT messages which tell the server **201** that the client **203** is going to sleep to wait for outstanding requests to be processed. The server sends a wake-up message so the client can complete its transactions.

As shown in portion **403**, when the client **203** receives a new request to perform a transaction, a test is performed to determine if the number of currently used credits is less than the credit limit. If so, the client updates its Credits Used number and issues a transaction message to the server **201**. When the client **203** sends the transaction message, the client includes a "hint" about the number of transactions that are pending at the client **203** (the Pending Count). Transactions become pending when the number of transaction requests exceeds the client's Credit Limit. The hints may be sent using an 8 bit field in the transaction message. If there are more than 2^8 requests pending, the field may be set to **255**. As described in conjunction with FIGUREs 5 and 6, the server **201** uses the hints to determine how to fairly allocate credits to clients.

In a resource constrained situation, the server may attempt to adjust the number of credits the client has, so it will send a "delta credit" message. The delta credit message contains the change in credits as a plus or minus number relative to what the server has currently allocated for the client to use. In this implementation, deltas are used rather than absolute values to prevent problems on networks where out of order receipt is possible. With this scheme it is possible for a client to use more credits than it is allowed. However, given enough time it will never be left in a state where this is perpetually true. If absolute numbers are used it is possible for a client to continually try to use more credits than it is allocated.

At portion **407**, if the client **203** receives a plus delta N credit message it can immediately send N additional requests to the server **201**. A credit acknowledge is sent as part of an existing request or as a separate message. If a separate message is used then only N-1 new requests are sent along with the acknowledge message. At portion **409**, if a minus delta credit message is received the client **203** waits until enough pending transactions have completed that sending a new request will not cause the new credit limit to be exceeded.

In networks where receive buffers have to be posted before messages can be received by either a client or server, a method has to be devised to flush out posted buffers from the receive queue when a server wants to reduce a clients credits by N. The server can either send N minus one delta messages to the client to allow the client to reclaim buffers out of the receive queue followed by the client sending N minus one credit messages to the server to allow the server to reclaim its buffers from the receive queue. The server would not reply to the N minus one credit messages. In a second method used by LWIO the server sends one minus N credit message to the client followed by the client sending N minus one credit messages to the server to allow the server to reclaim N buffers. The server will reply to each minus one credit message to allow the client to reclaim its buffers from the receive queue. This technique is used by LWIO running on VI networks which requires receive buffers to be posted.

Information Maintained by the Server

FIGURE 5 is a functional block diagram generally illustrating a server **201** configured for use in the network environment of FIGURE 2. The server **201** is responsible for handling client transaction requests and receives a connection **510** from each client. Each client can have more than one file open on a given connection **510**. In this particular implementation, credits are allocated to clients on a per connection basis but may be based on the number of files open or on the number of connections.

The server maintains server-side information **221** about each connection to assist the dynamic rebalancing mechanism of the invention. The server **201** stores the following information for each connection instance "i" (1) the number of outstanding requests from the client (Credits Used (CU_i)), (2) the maximum number of credits the client connection is allowed to use (Credit Limit (CL_i)), (3) the maximum hint sent by the client for a given credit timeout period (Client Pending Queue Count (CP_i)), and (4) the number of files currently open for use by the client connection (CF_i). The server **201** may also maintain information about acknowledge flags and the like.

FIGURE 6 is a pseudo-code representation of an illustrative process performed at the server **201** for interacting with the client **203** to achieve dynamic rebalancing of the client's allocated resources. The essential functionality of this pseudo-code could be implemented in software components executing on the server **201**.

The pseudo-code shown in FIGURE 6 is divided into four portions: a new client connection (portion **603**), a new client request (portion **605**), a credit message (portion **607**), and a timer expired (portion **609**). First, as mentioned above, when a client connects to the server **201** (at portion **603**), the server **201** and client negotiate a minimum number of credits for the client. If the credits are available, the server accepts the connection. Otherwise, the server **201** can reject the connection. It should be noted that if, during the life of a connection, a client exceeds the number of negotiated requests, the server may determine the appropriate reaction. Options include ignoring the new request, terminating the connection between the client and server, or simply processing the excessive requests.

As mentioned above, each time a client issues to the server a transaction request, the client includes a hint (e.g., the client's CP) about how many resources the client desires. Accordingly, at portion **605** the server **201** stores the client's CP in the server-side information **221**. The credit rebalancing does not occur unless a client has sent a hint to the server indicating its pending wait queue is greater than one. Once a client request is received with a pending count greater than one, rebalancing may occur at the expiration of a "credit timer." If another hint is received before the credit timer expires and the hint is larger than the currently-stored hint the new one is stored.

In this implementation, the timeout is global to all connections and is used to prevent excessively frequent changes in credit allocations. The timer will continue to be reset until all connections have acknowledged their credit messages. When the credit timeout expires, portion **609** of the process is invoked. The credit timeout is user configurable with the default being 2 seconds.

When the timer expires, the process first determines several things: the total number of existing connections (TC), the total number of files open (TF = sum of the Number of Files Open for all connections), the total credits being requested (TR = sum of the Client Pending Queue Length for all connections), and how many credits are currently in use (TU = Sum of Credits Used for all connections).

Rebalancing credits only occurs if the (TU + TR) is greater than the maximum number of credits available (MC), meaning the server **201** is in a resource constrained condition. The maximum number of credits available may be a function of the networking device being used and/or user specified values. If the server is not currently resource constrained, positive credit messages giving clients CP_i credits will be sent if a client connection has queued transactions ($CP_i > 0$) indicating it needs more credits. Credit messages might not be sent to connections that have not acknowledged their last credit message.

If (TU + TR) is greater than MC, then the new number of credits for each connection is computed. In this implementation, either of two polices are used for the rebalancing, one based on files or one based on connections. The choice depends on whether the server is configured to be fair to connections or files. If the policy is based

on files, then the new credit limit of a connection is computed as $NCL_i = MC * (CF_i / TF)$ where "NCL" is New Client Limit. If the policy is based on connections, then the new target is $NCL_i = MC / TC$. Priority can be introduced by assigning priority to each file ($FPRI_{i,j}$ in the rage of 1 to 32) or connection ($CPRI_i$ in the rage of 1 to 32). Index "i,j" is connection "i" file "j." The larger the value, the higher the priority.

Weighting using file priority could be used to give connections with higher priority files more credits as follows; $NCL_i = MC\frac{\sum_j FPRI_{i,j}}{\sum_{i,j} FPRI_{i,j}}$.

Weighting using connection priority could be used to give connection with higher priority more credits as follows; $NCL_i = MC\frac{CPRI_i}{\sum_i CPRI_i}$.

If CL_i for a particular connection is greater than NCL_i (meaning that the client is using more than its fair share of credits) and $(TU + TR) > MC$ then that particular client will get a negative credit delta message to release $N = (CL_i - NCL_i)$ credits. The server can send a single "minus N" delta credit message or N "minus one" delta credit messages. How the client acknowledges the message is detailed in the client section.

If the server is rebalancing credits and $((CL_i * \text{Completion Factor}) - CU_i) > 0$ (meaning that the client is not using all its credits) then the client will be sent a negative credit delta message to release $N = (CL_i - CU_i)$ credits. The server can send a single "minus N" delta credit message or N "minus one" delta credit messages. How the client acknowledges the message is detailed in the client section. The completion factor is used to control how many credits are reclaimed even if not used. This is intended to prevent clients that are in transition from having their credits taken prematurely.

If CL_i for a connection is less than NCL_i and CP_i is greater than 0 (meaning that the client has fewer than its fair share) a positive delta credit message will be sent to increase the client's credits by $N = \min(CP_i, NCL_i - CL_i, MC - TU)$. The number of credits used (TU) is then increased by N, and the number of credits being requested (TR) is reduced by N. How the client acknowledges the message is detailed above in the client section.

In order to prevent a situation where a client connection request is rejected due to all credits being used, a certain number of credits can be reserved for new connections. When a new connection is made the credit timer is set to allow redistribution of credits in a credit constrained environment. The number of reserved credits is configurable. When the server receives a credit message from the client it may respond with an acknowledgement.

In this manner, a server in a network environment can dynamically reallocate or rebalance its resources to fairly share those resources among each of a plurality of clients. As has been described, the invention enables a mechanism that grants as many resources to each requesting client as that client needs in a non-resource constrained situation, but rebalances the resources so that each client has its own fair share if the server's resources become constrained.

The above specification, examples and data provide a complete description of the concepts and illustrative implementations of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.